| Class Name: handler(from login.js) | |
| --- | --- |
| Parent Class: None  Subclass: None | |
| Responsibilities   * Handle HTTP POST requests for user login * Extract and log the user's email and password from the request body * Connect to the MongoDB database * Retrieve user information from the database using the provided email * Validate the provided password against the stored hashed password * Return a success message and user data (excluding the password) if authentication is successful * Return appropriate error messages if the user is not found or the password is invalid * Restrict the allowed HTTP methods to POST | Collaborators:   * clientPromise: For establishing a connection to the MongoDB database * bcrypt: For comparing the provided password with the hashed password stored in the database * req (Request object): To access the incoming request data * res (Response object): To send back the appropriate HTTP responses |

| Class Name: handler(from route.js) |
| --- |
| Parent Class: None  Subclass: None |
| Responsibilities  • Handle HTTP POST requests for user registration  • Connect to MongoDB database using clientPromise  • Check if the user already exists in the database4  • Hash the user's password using bcrypt  • Insert new user data into the database  • Return appropriate HTTP status codes and messages based on the outcome |
| Collaborator: NextResponse from 'next/server'  • clientPromise from '../../../lib/mongodb'  • bcrypt for password hashing |

| Class Name: handler(from route.js) |
| --- |
| Parent Class: None  Subclass: None |
| Responsibilities  • Handle HTTP POST requests for user registration  • Connect to MongoDB database using clientPromise  • Check if the user already exists in the database4  • Hash the user's password using bcrypt  • Insert new user data into the database  • Return appropriate HTTP status codes and messages based on the outcome |
| Collaborator: NextResponse from 'next/server'  • clientPromise from '../../../lib/mongodb'  • bcrypt for password hashing |

| Class Name: CurrentProjects(from CurrentProjects.js) |  |
| --- | --- |
| Parent Class: None  Subclass: None |  |
| Responsibilities   * Display Current Projects of user * Enhance UI purposes | Collaborators:   * None (MongoDB in the future) |

| Class Name: ConnectWith(Component) | |
| --- | --- |
| Parent Class: None  Subclass: None | |
| Responsibilities   * Create the user interface component as a container * Display the HTML heading with the title “Connect With” * Will display list of users | Collaborators:   * None (MongoDB in the future) |

| Class Name: LoginForm |
| --- |
| Parent Class: None  Subclass: None |
| Responsibilities  Render a login form for user authentication.  Manage form state for schoolEmail, password, and message.  Handle form submission and communicate with the server for user login.  Display messages to the user based on the login attempt result.  Redirect user to the home page upon successful login. |
| Collaborators:  useState  useRouter  setIsRegister |
|  |

| Class Name: QuickActions (from QuickActions.js) |  |
| --- | --- |
| Parent Class: None  Subclass: None |  |
| Responsibilities   * Allows user to click:   + Quick Actions   + Connect   + Create Project   + View Projects   + My Profile   + Notifications | Collaborators:   * None (MongoDB in the future) * Interaction with Other Classes (Future; Sprint 2) |

| Class Name: RegisterForm(Component) | |
| --- | --- |
| Parent Class: None  Subclass: None | |
| Responsibilities   * Manage our states for the user input fields which we have (school email, password, confirm password) * Use NextJS to render the form with input fields * Validate the password and confirm password field is a match * Send the POST request to server with the user’s registration details from the fields then the server posts it into the mongodb * Based on the post status from mongodb, if successful then display the server response * The user can switch to login | Collaborators:   * Using react useState for managing the state of the user fields * useRouter for navigation * SetisRegister is a prop function used to navigate between login and register |

### **Description of System Interaction with the Environment**

#### **Dependencies and Assumptions:**

1. **Operating System**:
   * The system is designed to be OS-independent, meaning it can run on Windows, macOS, and Linux. However, development and testing are typically done on a Unix-like environment (Linux or macOS).
2. **Programming Languages**:
   * **JavaScript**: For both client-side (React.js) and server-side (Node.js) development.
   * **HTML/CSS**: For client-side rendering and styling.
3. **Frameworks and Libraries**:
   * **Frontend**:
     + **React.js**: For building the user interface.
     + **React Router**: For handling routing in the single-page application.
     + **Axios**: For making HTTP requests from the client to the server.
   * **Backend**:
     + **Express.js**: For building the server-side RESTful APIs.
     + **Mongoose**: For interacting with MongoDB.
     + **Bcrypt.js**: For password hashing.
     + **Jsonwebtoken**: For user authentication.
     + **Cors**: For enabling Cross-Origin Resource Sharing.
   * **Development Tools**:
     + **Nodemon**: For automatic server restarts during development.
     + **Concurrently**: For running both the client and server concurrently during development.
4. **Database**:
   * **MongoDB**: Used to store user data, including emails, hashed passwords, and profile information.
5. **Network Configuration**:
   * The system requires a stable internet connection for accessing the MongoDB Atlas database.
   * Proper network configurations and permissions are needed, including IP whitelisting for MongoDB Atlas.

#### **Architecture Diagram**

+-------------------------------------------------------------+

| Frontend |

| +---------------------+ +-------------------------+ |

| | Register Component | | Login Component | |

| | | | | |

| | | | | |

| +---------+-----------+ +------------+------------+ |

| | | |

| | HTTP Request (Axios) | |

| v v |

| +---------------------+ +-------------------------+ |

| | Profile Component | | Local Storage | |

| | | | (JWT Token) | |

| | | | | |

| +---------------------+ +-------------------------+ |

+-------------------------------------------------------------+

^ ^

| |

| |

v v

+-------------------------------------------------------------+

| Backend |

| +---------------------+ +-------------------------+ |

| | Express.js | | MongoDB | |

| | | | | |

| | | | User Collection | |

| +---------+-----------+ +------------+------------+ |

| | | |

| | Route Handling | |

| v v |

| +---------------------+ +-------------------------+ |

| | Auth Routes | | | |

| | (Register, Login) | | | |

| +---------+-----------+ +-------------------------+ |

| | |

| | Profile Routes |

| v |

| +---------------------+ |

| | Profile Controller | |

| +---------------------+ |

| |

| +---------------------+ |

| | Middleware (JWT Auth)| |

| +---------------------+ |

+-------------------------------------------------------------+

<https://www.youtube.com/watch?v=7CqJlxBYj-M>

<https://www.youtube.com/watch?v=korRfKTDoxE>

REact:

<https://www.youtube.com/watch?v=CgkZ7MvWUAA&t=8678s>

NExtJS:  
<https://www.youtube.com/watch?v=GowPe3iiqTs&t=15338s>

<https://docs.google.com/document/d/144-8JrgqWrPEXMoKv4p5X9Fdg265JxSKA23ScJ4b68k/edit>